Intel® C++ Composer XE 2013 for Windows* Installation Guide and Release Notes

Document number: 321414-004US
4 October 2012

Table of Contents

1 Introduction

This document describes how to install the product, provides a summary of new and changed product features and includes notes about features and problems not described in the product documentation.

1.1 Change History

This section highlights important from the previous product version and changes in product updates. For information on what is new in each component, please read the individual component release notes.

1.1.1 Update 1 (2013.1)

- Intel® C++ Compiler XE 13.0.1
- Intel® Math Kernel Library 11.0 Update 1
- Intel® Integrated Performance Primitives 7.1 Update 1
- Intel® Threading Building Blocks 4.1 Update 1
- core_4th_gen_avx added for manual cpu dispatch
- Inline assembly and intrinsic support for Intel architecture code named Broadwell
- Support for Microsoft* loop pragma

- /Qcheck-pointers:w
- Corrections to reported problems

1.1.2 Changes since Intel® C++ Composer XE 2011

- Intel® C++ Compiler updated to version 13.0.
- Intel® Parallel Debugger Extension has been removed.
- Intel® Math Kernel Library updated to version 11.0
    - Removed support for Intel® Pentium® III processor. See the Knowledge Base article on Deprecations for further information.
- Intel® Integrated Performance Primitives updated to version 7.1
    - Intel® IPP static threaded libraries now available in separate package
- Intel® Threading Building Blocks updated to version 4.1
- Microsoft Windows Vista* and Windows Server 2003* are not supported.
- Microsoft Visual Studio 2005* is not supported.
- The Intel® Software Manager has been added to help you manage product updates and license activation
- New C++11 features
- Improved support for future Intel processors
- New Intel Performance Wizard
- Out-of-bounds memory checking
- Static Analysis Improvements

1.2 Product Contents

*Intel® C++ Composer XE 2013 Update 1 for Windows** includes the following components:

- Intel® C++ Compiler XE 13.0.1 for building applications that run on IA-32 or Intel® 64 architecture systems running the Windows* operating system
- Intel® Integrated Performance Primitives 7.1 Update 1
- Intel® Math Kernel Library 11.0 Update 1
- Intel® Threading Building Blocks 4.1 Update 1
- Integration into Microsoft* development environments
- Sample programs
- On-disk documentation

1.3 System Requirements

For an explanation of architecture names, see http://intel.ly/q9JVjE

- A PC based on an IA-32 or Intel® 64 architecture processor supporting the Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions (Intel® Pentium® 4 processor or later), or compatible non-Intel processor
    - For the best experience, a multi-core or multi-processor system is recommended
- 1GB RAM (2GB recommended)
- 4GB free disk space for all product features and all architectures

- Microsoft Windows XP*, Microsoft Windows 7*, Microsoft Windows 8*, Microsoft Windows Server 2008*, Microsoft Windows HPC Server 2008*, or Microsoft Windows Server 2012* (embedded editions not supported)
    - Microsoft Windows Server 2008 or Windows HPC Server 2008 requires Microsoft Visual Studio 2010* or Visual Studio 2008* SP1.
    - On Microsoft Windows 8 and Microsoft Windows Server 2012, the product installs into the "Desktop" environment. Development of "Windows 8 UI" applications is not supported.
- To use the Microsoft Visual Studio development environment or command-line tools to build IA-32 or Intel® 64 architecture applications, one of:
    - Microsoft Visual Studio 2012* Standard Edition (or higher edition) with C++ component installed
    - Microsoft Visual Studio 2010* Standard Edition (or higher edition) with C++ and "X64 Compiler and Tools" components installed [1]
    - Microsoft Visual Studio 2008* Standard Edition (or higher edition) with C++ and "X64 Compiler and Tools" components installed [1]
- To use command-line tools only to build IA-32 architecture applications, one of:
    - Microsoft Visual C++ Express 2012 for Windows Desktop*
    - Microsoft Visual C++ 2010* Express Edition
    - Microsoft Visual C++ 2008* Express Edition
- To use command-line tools only to build Intel® 64 architecture applications, one of:
    - Microsoft Visual C++ Express 2012 for Windows Desktop*
    - Microsoft Windows Software Development Kit Update for Windows 7* and .NET Framework 4*
- To read the on-disk documentation, Adobe Reader* 7.0 or later

Notes:

1. Microsoft Visual Studio 2008 Standard Edition installs the "x64 Compiler and Tools" component by default – the Professional and higher editions require a "Custom" install to select this. Microsoft Visual Studio 2010 includes x64 support by default.
2. The default for the Intel® compilers is to build IA-32 architecture applications that require a processor supporting the Intel® SSE2 instructions - for example, the Intel® Pentium® 4 processor. A compiler option is available to generate code that will run on any IA-32 architecture processor. However, if your application uses Intel® Integrated Performance Primitives or Intel® Threading Building Blocks, executing the application will require a processor supporting the Intel® SSE2 instructions.
3. Applications can be run on the same Windows versions as specified above for development. Applications may also run on non-embedded 32-bit versions of Microsoft Windows earlier than Windows XP, though Intel does not test these for compatibility. Your application may depend on a Win32 API routine not present in older versions of Windows. You are responsible for testing application compatibility. You may need to copy certain run-time DLLs onto the target system to run your application.

1.3.1 IA-64 Architecture (Intel® Itanium®) Development Not Supported

This product version does not support development on or for IA-64 architecture (Intel® Itanium®) systems. The version 11.1 compiler remains available for development of IA-64 architecture applications.

1.3.2 Windows Server 2003* and Windows Vista* Not Supported

Support has been removed for installation and use on Windows Server 2003 and Windows Vista. Intel recommends migrating to a newer version of these operating systems.

1.3.3 Visual Studio 2005* Not Supported

Support has been removed for installation and use with Visual Studio 2005. Intel recommends migrating to a newer version of Visual Studio*.

1.4 Documentation

Product documentation can be found in the `Documentation` folder as shown under Installation Folders.

Optimization Notice

Intel's compilers may or may not optimize to the same degree for non-Intel microprocessors for optimizations that are not unique to Intel microprocessors. These optimizations include SSE2, SSE3, and SSSE3 instruction sets and other optimizations. Intel does not guarantee the availability, functionality, or effectiveness of any optimization on microprocessors not manufactured by Intel. Microprocessor-dependent optimizations in this product are intended for use with Intel microprocessors. Certain optimizations not specific to Intel microarchitecture are reserved for Intel microprocessors. Please refer to the applicable product User and Reference Guides for more information regarding the specific instruction sets covered by this notice.

Notice revision #20110804

1.5 Samples

Samples for each product component can be found in the `Samples` folder as shown under Installation Folders.

1.6 Japanese Language Support

Intel compilers provide support for Japanese language users. Error messages, visual development environment dialogs and some documentation are provided in Japanese in addition to English. By default, the language of error messages and dialogs matches that of your operating system language selection. Japanese-language documentation can be found in the `ja_JP` subdirectory for documentation and samples.

Japanese language support will be available in an update on or after the release of Intel® C++ Composer XE 2013.

If you wish to use Japanese-language support on an English-language operating system, or English-language support on a Japanese-language operating system, you will find instructions at http://intel.ly/oZjpZs

1.7 Technical Support

If you did not register your compiler during installation, please do so at the Intel® Software Development Products Registration Center. Registration entitles you to free technical support, product updates and upgrades for the duration of the support term.

For information about how to find Technical Support, Product Updates, User Forums, FAQs, tips and tricks, and other support information, please visit http://www.intel.com/software/products/support/

Note: If your distributor provides technical support for this product, please contact them for support rather than Intel.

2 Installation

2.1 Intel® Software Manager

The installation now provides an Intel® Software Manager to provide a simplified delivery mechanism for product updates and provide current license status and news on all installed Intel® software products.

You can also volunteer to provide Intel anonymous usage information about these products to help guide future product design. This option, the Intel® Software Improvement Program, is not enabled by default – you can opt-in during installation or at a later time, and may opt-out at any time. For more information please see http://intel.ly/SoftwareImprovementProgram.

2.2 Pre-installation Steps

2.2.1 Configure Visual Studio for 64-bit Applications

If you are using Microsoft Visual Studio 2008* and will be developing 64-bit applications (for the Intel® 64 architecture) you may need to change the configuration of Visual Studio to add 64-bit support.

If you are using Visual Studio 2008* Standard Edition, or Visual Studio 2010* Professional Edition or higher, no configuration is needed to build Intel® 64 architecture applications. For other editions:

1. From Control Panel > Add or Remove Programs, select “Microsoft Visual Studio 2008” > Change/Remove. The Visual Studio Maintenance Mode window will appear. Click Next.
2. Click Add or Remove Features
3. Under “Select features to install”, expand Language Tools > Visual C++
4. If the box “X64 Compiler and Tools” is not checked, check it, then click Update. If the box is already checked, click Cancel.

Note that Visual C++ Express Edition does not support 64-bit development.

2.3 Installation

The installation of the product requires a valid license file or serial number. If you are evaluating the product, you can also choose the "Evaluate this product (no serial number required)" option during installation.

If you received your product on DVD, insert the first product DVD in your computer's DVD drive; the installation should start automatically. If it does not, open the top-level folder of the DVD drive in Windows Explorer and double-click on setup.exe.

If you received your product as a downloadable file, double-click on the executable file (.EXE) to begin installation. Note that there are several different downloadable files available, each providing different combinations of components. Please read the download web page carefully to determine which file is appropriate for you.

You do not need to uninstall previous versions or updates before installing a newer version – the new version will coexist with the older versions

2.3.1 Changes to system PATH may cause temporary in-operation of command shell (cmd.exe)

On Windows* 7 or 8, if the installation's additions to the system PATH cause the PATH length to consist of between 2000-4000 characters, this could cause the Windows command prompt (cmd.exe) to not work until the next reboot. If you observe such behavior after installation, reboot and if the symptom persists, contact Technical Support.

2.3.2 Silent Install

For information on automated or "silent" install capability, please see http://intel.ly/nKrzhv

2.3.3 Cluster Installation

If Microsoft Compute Cluster Pack* is present, and the installation detects that the installing system is a member of a cluster, the product will be installed on all visible nodes of the cluster when a "Full" installation is requested. If a "Custom" installation is requested, you will be given the option to install on the current node only.

2.3.4 Using a License Server

If you have purchased a "floating" license, see http://intel.ly/pjGfwC for information on how to install using a license file or license server. This article also provides a source for the Intel® License Server that can be installed on any of a wide variety of systems.

2.4 Changing, Updating and Removing the Product

Use the Windows Control Panel "Add or Remove Products" applet to change which product components are installed or to remove the product.

When installing an updated version of the product, you do not need to remove the older version first. You can have multiple versions of the compiler installed and select among them. If you

remove a newer version of the product you may have to reinstall the integrations into Microsoft Visual Studio from the older version.

2.5 Installation Folders

The installation folder arrangement is shown in the diagram below. Not all folders will be present in a given installation.

- C:\Program Files\Intel\Composer XE 2013
  - bin
    - ia32
    - ia32_intel64
    - intel64
    - sourcechecker
  - compiler
    - include
      - cilk
      - ia32
    - lib
      - ia32
      - intel64
    - perf_headers
  - Documentation
    - en_US
      - compiler_c
        - cl
      - gs_resources
      - ipp
        - get_started_files
        - ipp_userguide
        - tutorials
      - mkl
        - get_started_files
        - mkl_userguide
        - tutorials
      - ssadiag_docs
      - tbb
        - get_started_files
        - html
        - tutorial
      - tutorials
        - cmp_gap_c
        - cmp_thd_c
        - cmp_vec_c

- msvhelp
  - 1033
    - compiler_c
    - ipp
    - mkl
    - ssadiag
    - tbb
  - vshelp
    - intel.cppprodocs
    - intel.cprocompilerdocs
    - intel.ippdocs
    - intel.mkldocs
    - intel.sssadiag
    - intel.tbbdocs
- Help
- ipp
  - bin
  - demo
  - include
  - interfaces
  - lib
  - tools
- mkl
  - benchmarks
  - bin
  - examples
  - include
  - interfaces
  - lib
  - tests
  - tools
- redist
  - ia32
    - compiler
      - 1033
      - irml
      - irml_c
    - ipp
      - 1033
    - mkl
      - 1033
    - tbb
      - vc_mt

        - vc8
        - vc9
        - vc10
    - intel64
      - compiler
        - 1033
        - irml
        - irml_c
      - ipp
        - 1033
      - mkl
        - 1033
      - tbb
        - vc_mt
        - vc8
        - vc9
        - vc10
- Samples
  - en_US
    - C++
    - ipp
- tbb
  - bin
  - examples
  - include
    - serial
      - tbb
    - tbb
      - compat
      - internal
      - machine
  - lib
    - ia32
      - vc_mt
      - vc8
      - vc9
      - vc10
    - intel64
      - vc_mt
      - vc8
      - vc9
      - vc10
- VS Integration

        - C++
            - VS2008

Where the folders under `bin`, `include` and `lib` are used as follows:

- `ia32`: Files used to build applications that run on IA-32
- `intel64`: Files used to build applications that run on Intel® 64
- `ia32_intel64`: Compilers that run on IA-32 to build applications that run on Intel®64

If you are installing on a system with a non-English language version of Windows, the name of the `Program Files` folder may be different. On Intel® 64 architecture systems, the folder name is `Program Files (X86)` or the equivalent.

By default, updates of a given version will replace the existing directory contents. When the first update is installed, the user is given the option of having the new update installed alongside the previous installation, keeping both on the system. If this is done, the top-level folder name for the older update is changed to `Composer XE 2013.`*`nnn`* where *`nnn`* is the update number.

3 Intel® C++ Compiler

This section summarizes changes, new features and late-breaking news about the Intel C++ Compiler.

3.1 Compatibility

In version 11, the IA-32 architecture default for code generation has changed to assume that Intel® Streaming SIMD Extensions 2 (Intel® SSE2) instructions are supported by the processor on which the application is run. See below for more information.

3.2 New and Changed Features

C++ Composer XE 2013 now contains Intel® C++ Compiler XE 13.0. The following features are new or significantly enhanced in this version. For more information on these features, please refer to the documentation.

- Improved support for 3rd Generation Intel® Core™ processor family (/QxCORE-AVX-I and /QaxCORE-AVX-I) and future Intel processors supporting Intel® Advanced Vector Extensions 2 (Intel® AVX2) (/QxCORE-AVX2 and /QaxCORE-AVX2)
- New Intel Performance Wizard for getting started in identifying optimal compiler option settings (in Microsoft Visual Studio*, under “Intel Composer XE 2013->Start Performance Wizard”)
- Features from C++11 (/Qstd:c++0x)
    - Additional type traits
    - Uniform initialization
    - Generalized constant expressions (partial support)
    - noexcept
    - Range based for loops

    - Conversions of lambdas to function pointers
    - Implicit move constructors and move assignment operators
- Out-of-bounds memory checking (/Qcheck-pointers)
- Static Analysis improvements:
    - More options for controlling the analysis for Static Analysis (/Qdiag-enable:sc-{full|concise|precise)
    - Improvements to Windows* one-click interface
    - New progress bar
    - Support for Microsoft* SAL language extensions

3.2.1 New versions of Microsoft Visual Studio*

Microsoft Visual Studio 2012* is now supported for the development of desktop applications. Development of "Windows 8 UI" applications is not supported.

3.2.2 core_4th_gen_avx added for manual cpu dispatch in Composer XE 2013 Update 1

The cpuid "`core_4th_gen_avx`" is now supported for use with the `cpu_dispatch` and `cpu_specific` manual cpu dispatch mechanisms. This cpuid targets processors that support Intel® Advanced Vector Extensions 2 (Intel® AVX2).

3.2.3 Support for Microsoft loop pragma syntax added to Composer XE 2013 Update 1

Support for the Microsoft Visual C++ 2012* compiler's `#pragma loop [hint_parallel(n),no_vector,ivdep]` is added for Composer XE 2013 Update 1.

3.2.4 Inline assembly and intrinsic support for Intel architecture code named Broadwell added to Composer XE 2013 Update 1

Some new instructions have been added in the upcoming Intel architecture code named Broadwell. Composer XE 2013 Update 1 has added inline assembly and intrinsic support for these instructions. Intrinsics are defined in `immintrin.h`.

```
extern int _rdseed16_step(unsigned short *random_val);
extern int _rdseed32_step(unsigned int *random_val);
extern int _rdseed64_step(unsigned __int64 *random_val);
```

These intrinsics generate random numbers of 16/32/64 bit wide random integers. These intrinsics are mapped to the hardware instruction `RDSEED`. The generated random value is written to the given memory location and the success status is returned - 1if the hardware returned a valid random value, and 0 otherwise.

The difference between `rdseed` and `rdrand` intrinsics is that `rdseed` intrinsics meet the NIST SP 800-90B and NIST SP 800-90C standards, while the `rdrand` meets the NIST SP 800-90A standard.

```
extern unsigned char _addcarry_u32(unsigned char c_in, unsigned int
src1, unsigned int src2, unsigned int *sum_out);
```

```
extern unsigned char _addcarry_u64(unsigned char c_in, unsigned
__int64 src1, unsigned __int64 src2, unsigned __int64 *sum_out);
```

The intrinsic computes the sum of two 32/64 bit wide integer values (`src1, src2`) and a carry-in value. The carry-in value is considered 1 for any non-zero `c_in` input value or 0 otherwise. The sum is stored to a memory location referenced by `sum_out` argument:

```
*sum_out = src1 + src2 + (c_in !=0 ? 1 : 0)
```

The intrinsic does not perform validness check of a memory address pointed by `sum_out` thus it cannot be used to find out if a sum produces carry-out without storing result of the sum. The return value of the intrinsic is a carry-out value generated by sum. The sum result is stored into memory location pointed by `sum_out` argument.

```
extern unsigned char _subborrow_u32(unsigned char b_in, unsigned int
src1, unsigned int src2, unsigned int *diff_out);

extern unsigned char _subborrow_u64(unsigned char b_in, unsigned
__int64 src1, unsigned __int64 src2, unsigned __int64 *diff_out);
```

The intrinsic computes the sum of a 32/64 bit wide unsigned integer value `src2` and a borrow-in value and then subtracts the result of the sum from the 32/64 bit wide unsigned integer value `src1`. The borrow-in value is considered 1 for any non-zero `b_in` input value or 0 otherwise. The difference is then stored to a memory location referenced by `diff_out` argument:

```
*diff_out = src1 + (src2 + (b_in !=0 ? 1 : 0))
```

The intrinsic does not perform validness check of a memory address pointed by `diff_out` thus it cannot be used to find out if a subtraction produces borrow-out without storing the result of the subtraction. The return value of the intrinsic is a borrow-out value generated by subtraction. The result of the subtraction is stored into memory location pointed by the `diff_out` argument.

```
extern unsigned char _addcarryx_u32(unsigned char c_in, unsigned int
src1,   unsigned int src2, unsigned int *sum_out);

extern unsigned char _addcarryx_u64(unsigned char c_in, unsigned
__int64 src1,   unsigned __int64 src2, unsigned __int64 *sum_out);
```

The intrinsic computes sum of two 32/64 bit wide integer values (`src1, src2`) and a carry-in value. The carry-in value is considered 1 for any non-zero `c_in` input value or 0 otherwise. The sum is stored to a memory location referenced by `sum_out` argument:

```
*sum_out = src1 + src2 + (c_in !=0 ? 1 : 0)
```

The intrinsic does not perform validness check of a memory address pointed by `sum_out` thus it cannot be used to find out if a sum produces carry-out without storing the result of the sum.

The intrinsic is translated to either a `ADCX` or `ADOX` instruction depending on compiler's decision. By their design these instructions allow running of two interleaved add-with-carry instruction sequences in parallel via using `ADCX` and `ADOX` instructions for these sequences respectively. The return value of the intrinsic is the carry-out value generated by the sum. The sum result is stored into memory location pointed by the `sum_out` argument.

New `_MM_HINT_ET0` hint to `_mm_prefetch` instrinsic

The `_MM_HINT_ET0` hint makes the intrinsic being lowered to the instruction `PREFETCHW` which is supported by the Intel architecture code name Broadwell. Check if the target CPU supports the instruction `PREFETCHW` before using `_MM_HINT_ET0`.

3.2.5 Intel® Cilk™ Plus "scalar" Clause removed

The "scalar" clause used optionally with Intel® Cilk™ Plus elemental functions is removed in this release. Please use the functionally equivalent "uniform" clause instead.

3.2.6 Support for Intel® Advanced Vector Extensions 2 (Intel® AVX2) Instructions in 2011 Update 7

The compiler in Intel® C++ Composer XE 2011 Update 7 supports use of Intel® AVX2 instructions in inline assembly and in intrinsics in immintrin.h.

3.2.7 Intel® Cilk™ Plus Array Notations Semantics Change in 2011 update 6

In Intel® C++ Composer XE 2011, an Intel® Cilk™ Plus array section assignment like the following:

```
a[:] = b[:] + c[:];
```

could potentially generate temporary copies of the results, impacting performance.

Starting in Intel® C++ Composer XE 2011 update 6, if an array section on the right hand side of an assignment (in the example given, b[:] or c[:]) partially overlaps the array section on the left hand side (in the example given, a[:]) in memory, this assignment will be undefined, and it is up to the programmer to assure that there is no partial overlap in memory on assignments in order to get defined behavior.

An exception to this is if the array sections completely overlap, for example:

```
a[:] = a[:] + 3;
```

Since array a overlaps itself completely, this summation will work as expected.

3.2.8 Additional Keywords for `/Qsox` option, default changed in 2011 update 3

The `/Qsox` option, which adds information to the object file about compiler options used and procedure profiling information, has been enhanced to let the user request that the list of inlined functions be included and to let the user exclude information about procedure profiling.

The syntax for `/Qsox` is now:

`/Qsox[-]`
`/Qsox=`*`keyword`*`[,`*`keyword`*`]`

Where *`keyword`* is one of `inline` or `profile`. If `/Qsox` is specified with no keywords, only the command line options are included – this is a change from previous releases. To maintain the previous behavior, use `/Qsox=profile`. Multiple `/Qsox` options may be specified on the command line – if so, they are interpreted in left-to-right order.

The information is added to the object file as comment directives. These are ignored by Microsoft linkers beginning with Visual Studio 2005, therefore the information will not be present in the executable.

3.2.9 Three intrinsics changed in 2011 update 2

Three intrinsics (_rdrand16_step(), _rdrand32_step(), _rdrand64_step()) have been changed in update 2. The documentation has not been updated with these new changes. These intrinsic return a hardware-generated random value and are declared in the “immintrin.h” header file.

These three intrinsics are mapped to a single RDRAND instruction, generate random numbers of 16/32/64 bit wide random integers.

Syntax

1. extern int _rdrand16_step(unsigned short *random_val);
2. extern int _rdrand32_step(unsigned int *random_val);
3. extern int _rdrand64_step(unsigned __int64 *random_val);

Description

> The intrinsics perform one attempt to generate a hardware generated random value using the instruction RDRAND. The generated random value is written to the given memory location and the success status is returned: 1 if the hardware returned a valid random value and 0 otherwise.

Return

> A hardware-generated 16/32/64 random value.

Constraints

> The _rdrand64_step() intrinsic can be used only on systems with the 64-bit registers support.

3.2.10 Static Analysis Feature (formerly “Static Security Analysis” or “Source Checker”) Requires Intel® Inspector XE

The “Source Checker” feature, from compiler version 11.1, has been enhanced and renamed “Static Analysis”. The compiler options to enable Static Analysis remain the same as in compiler version 11.1 (for example, `/Qdiag-enable:sc`), but the results are now written to a

file that is interpreted by Intel® Inspector XE rather than being included in compiler diagnostics output.

3.2.10.1 The command line utility "inspxe-runsc.exe" changed since 2011 update 2

This utility is distributed with Intel® Composer XE 2011 and has been changed since 2011 update 2. This change only affects users who use Composer XE 2011 to perform Static Analysis. Those that do not use Static Analysis and those that perform Static Analysis without using this utility are unaffected. Static Analysis is only available to users of Intel® Parallel Studio XE, Intel® C++ Studio XE, or Intel® Fortran Studio XE versions 2011 or 2013, so users who do not have those products are unaffected.

inspxe-runsc executes a **build specification,** a description of how an application is built. Usually build specification files are generated by observing a build as it executes and recoding the compilations and links that are performed. inspxe-runsc repeats these actions using the Intel compiler in Static Analysis mode. Static Analysis results are generated at the link step so a build specification that describes a build with more than one link step will generate more than one Static Analysis result when inspxe-runsc is invoked.

The versions of inspxe-runsc included in Composer XE 2011 and Composer XE 2011 Update 1 generate all the Static Analysis results in a single directory. In the multiple link case this violated the rule that all the Static Analysis results for one and only one project must be created in the same directory. The updated version of inspxe-runsc respects this rule by generating results for each link step in a separate directory. The name of that directory is formed from the name of the file being linked. Thus if a build specification describes a project that builds two executables, file1.exe and file2.exe, then earlier versions of inspxe-runsc would create two results, one for file1 and one for file2, say r000sc and r001sc, in the same directory. The new version of inspxe-runsc will also create two results, but the one for file1 will be created in "My Inspector XE results – file1\r000sc" and the one for file2 will be created in "My Inspector XE results – file2\r000sc". The directories containing the results are both created in the same parent directory.

Inspxe-runsc has a command line switch, -result-dir (-r), that specifies where results are to be created. The meaning of this switch has changed. Previous this would name the directory where the result itself, say r000sc, would be created. Now it names the parent directory where the "My Inspector XE Results - name" directory or directories will be created. So the directory named in the –r switch is effectively two levels up from the results themselves.

The change to inspxe-runsc effectively moves the result directory, and user action is required to adapt to this change. Those using scripts that invoke inspxe-runsc with the –r switch must update their scripts to reflect the new interpretation of the –r switch argument described earlier. Users must move their old result files into the new directory so that Static Analysis results produced by earlier versions of inspxe-runsc share the same directory as results produced by the new version of inspxe-runsc. Users that had been using inspxe-runsc with a build specification with only one link step should move their old results into a directory of the form "My Inspector XE results – name". If this is not done, then all the problems in the newly created result will appear to be "New". Users that had been using inspxe-runsc with a build specification

with multiple link steps have been having various issues with Static Analysis that will be resolved by using the new utility. Such users are best advised to copy the most recent into their old results into each of the new "My Inspector XE results – name" directories. This offers the best chance that some old problem state information will be correctly applied to new results when they are created in the future.

3.2.11 Intel® C++ Project File Compatibility

The Intel C++ project file (`.icproj`) format changed in version 13.0 (Composer XE 2013). If you open a project created with an earlier version of Intel C++, you will get a message indicating that the project needs to be converted. A version 13.0 project cannot be used by an earlier version of the Intel C++ integration but you can use older versions of the compiler that you have installed through `Tools > Options > Intel C++ > Compilers`.

3.3 New and Changed Compiler Options

For details on these and all compiler options, see the Compiler Options section of the on-disk documentation.

- /Qvec-report6
- /Qextend-arguments:[32|64]
- `/Qguide-profile:<[file|dir]>[,[file|dir],...]`
- `/Qcheck-pointers:<arg>`
- `/Qcheck-pointers-dangling:<arg>`
- `/Qcheck-pointers-undimensioned[-]`
- `/Qstd=c++11 (same as /Qstd=c++0x)`
- /check:<keyword>[,<keyword>,...]
- /Qdiag-enable:sc-{full|concise|precise}
- /Qdiag-enable:sc-single-file
- /Qdiag-enable:sc-enums
- /watch:<keyword>
- /nowatch
- /Qvc11
- /Qgcc-dialect:<version>
- /Qipp-link:{static|dynamic|static_thread}

For a list of deprecated compiler options, see the Compiler Options section of the documentation.

3.3.1 /Qcheck-pointers:w added to Composer XE 2013 Update 1

Functionality has been added in update 1 to perform write-only error checking in Pointer Checker.

3.3.2 /Qipp-link option

This option is used with /Qipp to indicate which variant of the Intel® Integrated Performance Primitives libraries should be used. There are three options, static to link against the static single-threaded libraries, dynamic to link against the dynamic libraries, or static-thread to link

against the static multithreaded libraries. Note that the static multithreaded libraries are only available in a separate package.

3.3.3 Deprecated Options

Use following method to find all the deprecated compiler options:

1) Open a command prompt from Start menu: Start > All Programs > Intel Parallel Studio XE 2013 > Command Prompt > Parallel Studio XE with Intel Compiler XE v13.0 [Update xx] > IA-32/Intel® 64 Visual Studio xxx mode
2) Run command:

```
>> icl /? deprecate
```

3.4 Other Changes

3.4.1 Build Environment Command Script Change

The command window script used to establish the build environment allows the optional specification of the version of Microsoft Visual Studio to use. If you are not using the predefined Start menu shortcut to open a build environment window, use the following command to establish the proper environment:

```
"<install-dir>\bin\compilervars.bat" arch [vs]
```

Where *arch* is one of followings as appropriate for the target architecture you want to build for:

- ia32
- ia32_intel64
- intel64

vs is optional and can be one of followings. If *vs* is not specified, the version of Visual Studio specified at installation time for command-line integration is used by default.

- vs2012
- vs2010
- vs2008

If you also have Intel® Visual Fortran Composer XE 2013 installed, this command will also establish the environment for using that compiler.

The script file names iclvars.bat and ifortvars.bat have been retained for compatibility with previous releases.

3.4.2 OpenMP* Legacy Libraries Removed

The OpenMP “legacy” libraries have been removed in this release. Only the “compatibility” libraries are provided.

3.4.3 OpenMP* Static Libraries Removed

The static versions of the OpenMP* runtime libraries have been removed from this release. You must link these runtimes dynamically.

3.4.4 Using Intel C++ Projects with a Source Control System

If your project is managed under a source control system, for example, Microsoft Visual Source Safe* or Microsoft Visual Studio Team Foundation Server*, there are additional steps you must follow in order to use the Intel C++ project system with your project. A detailed article on this topic is available at http://intel.ly/pImnpo

3.5 Known Issues

3.5.1 Compiler Known Issues

3.5.1.1 Command-Line Diagnostic Issue for Filenames with Japanese Characters

The filename in compiler diagnostics for filenames containing Japanese characters may be displayed incorrectly when compiled within a Windows command shell using the native Intel® 64 architecture compiler. It is not a problem when using Visual Studio or when using the Intel® 64 architecture cross-compiler or IA-32 architecture compiler

3.5.2 Visual Studio Known Issues

3.5.3 Showing Documentation Issue with Microsoft Visual Studio 2012* and Windows Server 2012*

If on Windows Server 2012* you find that you cannot display help or documentation from within Visual Studio 2012*, correcting a security setting for Microsoft Internet Explorer* usually corrects the problem. From `Tools > Internet Options > Security`, change the settings for `Internet Zone` to allow "`MIME Sniffing`" and "`Active Scripting`".

3.5.3.1 MSVCP90D.dll (or other Microsoft runtime DLL) is missing

There are situations where the sample projects provided (or any Microsoft Visual C++* project potentially) could run into a runtime System Error where the application cannot find a Microsoft Visual Studio* runtime DLL. This is related to manifest files and SXS assemblies potentially missing. The simplest solution is to go to your redist directory for the version of Microsoft Visual Studio* you are using (default location would be c:\program files[(x86)]\Microsoft Visual Studio X.X\VC\redist). There will be several subdirectories under this location, sorting files out by amd64, x86 or Debug_NonRedist (if you have D in the runtime name, this usually indicates a Debug library found in this folder). Find the appropriate folder that contains the runtime you are looking for, and then copy the entire contents of that folder (including a .manifest file) to the directory where the .exe you are trying to run is located.

3.5.3.2 Visual Studio 2010 sets default of `/fp:precise`

A project created in or converted to Visual Studio 2010 will have the command line option `/fp:precise` set by default. This option sets the "floating point model" to improve consistency for floating point operations by disabling certain optimizations, reducing performance. To set the

option back to the Intel default of `/fp:fast`, change the project property C++ > Optimization > Floating Point Model to Fast.

3.5.3.3 Language packs of Visual Studio 2010

If you install a new language pack of Visual Studio 2010 after installing the Intel C++ Composer XE 2013, you may not see the Intel C++ Compiler specific options in the Project Property dialog. Please try the following to fix the issue:

1) If directory "`<program files>\MSBuild\Microsoft.Cpp\v4.0\Platforms\[Win32|x64]\PlatformToolsets\Intel C++ Compiler XE 13.0\1033`" exists, copy all files to "`<program files>\MSBuild\Microsoft.Cpp\v4.0\Platforms\[Win32|x64]\PlatformToolsets\Intel C++ Compiler XE 13.0\<locale-ID>`".

2) If directory "`<program files>\MSBuild\Microsoft.Cpp\v4.0\Platforms\[Win32|x64]\PlatformToolsets\v100\1033\`" exists, copy all files to "`<program files>\MSBuild\Microsoft.Cpp\v4.0\Platforms\[Win32|x64]\PlatformToolsets\v100\<locale-ID>`".

* The `<locale-ID>` is the language pack.

Another method is to uninstall the Intel C++ Composer XE 2013 and reinstall the Intel C++ Composer XE 2013.

3.5.4 Intel® Cilk™ Plus Known Issues

- Microsoft C++ Structured Exception Handling (SEH) will fail if an SEH exception is thrown after a steal occurs and before the corresponding `_Cilk_sync`.

3.5.5 Guided Auto-Parallel Known Issues

Guided Auto Parallel (GAP) analysis for single file, function name or specific range of source code does not work when Whole Program Interprocedural Optimization (`/Qipo`) is enabled. The workaround is to disable `/Qipo` – in Visual Studio, this is `Project > [projectname] Properties > C++ > Optimization > Interprocedural Optimization > No`.

3.5.6 Static Analysis Known Issues

3.5.6.1 Excessive false messages on C++ classes with virtual functions

Note that use of the Static Analysis feature also requires the use of Intel® Inspector XE.

Static Analysis reports a very large number of incorrect diagnostics when processing any program that contains a C++ class with virtual functions. In some cases the number of spurious diagnostics is so large that the result file becomes unusable.

If your application contains this common C++ source construct, add the following command line switch to suppress the undesired messages: `/Qdiag-disable:12020,12040` (Windows) or `-diag-disable 12020,12040` (Linux). **This switch must be added at the link step because that is when static analysis results are created.** Adding the switch at the compile

step alone is not sufficient. In Microsoft Visual Studio, add this to the property page Linker > Command Line.

If you are using a build specification to perform static analysis, add the `-disable-id 12020,12040` switch to the invocation of the inspxe-runsc, for example,

```
inspxe-runsc -spec-file mybuildspec.spec -disable-id 12020,12040
```

If you have already created a static result that was affected by this issue and you are able to open that result in the Intel® Inspector XE GUI, then you can hide the undesired messages as follows:

- The messages you will want to suppress are `"Arg count mismatch"` and `"Arg type mismatch"`. For each problem type, do the following:
- Click on the undesired problem type in the Problem filter. This hides all other problem types.
- Click on any problem in the table of problem sets
- Type control-A to select all the problems
- Right click and select *Change State -> Not a problem* from the pop-up menu to set the state of all the undesired problems
- Reset the filter on problem type to All
- Repeat for the other unwanted problem type
- Set the Investigated/Not investigated filter to *Not investigated*. You may have to scroll down in the filter pane to see it as it is near the bottom. This hides all the undesired messages because the "Not a problem" state is considered a "not investigated" state.

4 Intel® Integrated Performance Primitives

This section summarizes changes, new features and late-breaking news about this version of Intel® Integrated Performance Primitives (Intel® IPP). For detailed information about IPP see the following links:

- **New features**: see the information below and visit the main Intel IPP product page on the Intel web site at: http://intel.ly/OG5IF7; and the Intel IPP Release Notes at http://intel.ly/OmWI4d.

- **Documentation, help, and samples**: see the documentation links on the IPP product page at: http://intel.ly/OG5IF7.

4.1 Intel® IPP static threaded Libraries are Available as a Separate Download

If you require the static threaded version of the Intel® IPP libraries, they are no longer provided in the default Composer XE package. There should be a separate package available from the same area where you downloaded the Composer XE package that contains these libraries.

4.2 Intel® IPP Cryptography Libraries are Available as a Separate Download

The Intel® IPP cryptography libraries are available as a separate download. For download and installation instructions, please read http://intel.ly/ndrGnR

4.3 Intel® IPP Code Samples

The Intel® IPP code samples are organized into downloadable packages at http://intel.ly/pnsHxc

The samples include source code for audio/video codecs, image processing and media player applications, and for calling functions from C++, C# and Java*. Instructions on how to build the sample are described in a readme file that comes with the installation package for each sample.

5 Intel® Math Kernel Library

This section summarizes changes, new features and late-breaking news about this version of the Intel® Math Kernel Library (Intel® MKL). All the bug fixes can be found here: http://intel.ly/OeHQqf

5.1 Notices

Please refer to the Knowledge Base article on Deprecations for more information on the following notices

- Microsoft Windows* System PATH environment variable is no longer set during installation
- Removed Intel MKL GNU Multiple Precision* (GMP) function interfaces
- Disabled timing function mkl_set_cpu_frequency() to perform useful work — use mkl_get_max_cpu_frequency(), mkl_get_clocks_frequency(), and mkl_get_cpu_frequency() as described in the Intel MKL Reference Manual
- Removed MKL_PARDISO constant — used MKL_DOMAIN_PARDISO to specify the PARDISO domain with the mkl_domain_set_num_threads() function
- Removed special backward compatibility functions for convolution and correlation functions in Intel MKL 10.2 update 4
- Removed the OpenMP* static runtime library from the Windows* version of Intel MKL and Intel® compilers
- Removed support for Intel® Pentium® III processor. The minimal supported instruction set will be Intel® Streaming SIMD Extensions 2 (Intel® SSE2).
- Documentation:
    - The Intel MKL Reference Manual in HTML format is no longer available with the product

5.2 Changes in This Version

5.2.1 What's New in Intel® MKL 11.0 Update 1

- BLAS:

    - Improved DGEMM and double-precision Level 3 BLAS performance on AMD* Family “Bulldozer” CPUs
- PARDISO: Imaginary part of the diagonal values for Hermitian matrices are ignored
- Cluster FFT:
    - Improved hybrid Cluster FFT (MPI + OpenMP*) performance up to 2 times
    - A new Cluster FFT algorithm (Segment of Interest FFT) that uses less communication was implemented for 1D FFTs and it can be enabled by setting the environment variable "MKL_CFFT_SOI_ENABLE" to "YES" or "1" — see more info in the Intel® MKL documentation
- VSL:
    - Added support of VSL_SS_METHOD_FAST_USER_MEAN method for computation of descriptive Summary Statistics estimates given user-provided mean
    - Improved performance of VSL_SS_METHOD_FAST method for computations of descriptive Summary Statistics estimates on Intel® Xeon® processor E5-2690 CPU
- Transposition: Improved performance of Out-of-place transposition on 2nd generation Intel® Core™ microarchitecture (up to 7x)
- Service functions: Removed seven service functions with obsolete names (see more details in this article on obsolete service functions removed)
- Bug fixes

5.2.2 Changes in Initial Release

- Conditional Bitwise Reproducibility (CBWR): New functionality in Intel MKL now allows you to balance performance with reproducible results by allowing greater flexibility in code branch choice and by ensuring algorithms are deterministic. See the Intel MKL User’s Guide for more information. Refer to the CBWR Knowledge Base Article for more information.
- Intel MKL also introduces optimizations using the new Intel® Advanced Vector Extensions 2 (AVX2) including the new FMA3 instructions. See the Knowledge Base article on support for Intel® AVX2
- BLAS:
    - Improved DSYRK/SSYRK performance for 64-bit programs supporting Intel® Advanced Vector Extensions (Intel® AVX)
- LAPACK:
    - Introduced support for LAPACK version 3.4.1
- FFT :
    - Added configuration parameter DFTI_THREAD_LIMIT which limits the number of threads per descriptor
    - Added support for 1D real-to-complex transforms with sizes given by 64-bit prime integers
- VML /VSL:
    - Improved performance of viRngGeometric on Intel® Advanced Vector Extensions (Intel AVX)

    - Implemented threading in Data Fitting Integrate1d function
- Transposition: Parallelized in-place transposition of square matrices with leading dimensions greater than the matrix size for single and double precisions improving its performance significantly
- Implemented local threading control function (mkl_set_num_threads_local) which increases flexibility in threading control
- Link Line Advisor:
    - Added Help-Me functionality for selecting architecture (IA-32/Intel® 64) and interface layer (LP64/ILP64)

5.3 Attributions

As referenced in the End User License Agreement, attribution requires, at a minimum, prominently displaying the full Intel product name (e.g. "Intel® Math Kernel Library") and providing a link/URL to the Intel® MKL homepage (http://www.intel.com/software/products/mkl) in both the product documentation and website.

The original versions of the BLAS from which that part of Intel® MKL was derived can be obtained from http://www.netlib.org/blas/index.html.

The original versions of LAPACK from which that part of Intel® MKL was derived can be obtained from http://www.netlib.org/lapack/index.html. The authors of LAPACK are E. Anderson, Z. Bai, C. Bischof, S. Blackford, J. Demmel, J. Dongarra, J. Du Croz, A. Greenbaum, S. Hammarling, A. McKenney, and D. Sorensen. Our FORTRAN 90/95 interfaces to LAPACK are similar to those in the LAPACK95 package at http://www.netlib.org/lapack95/index.html. All interfaces are provided for pure procedures.

The original versions of ScaLAPACK from which that part of Intel® MKL was derived can be obtained from http://www.netlib.org/scalapack/index.html. The authors of ScaLAPACK are L. S. Blackford, J. Choi, A. Cleary, E. D'Azevedo, J. Demmel, I. Dhillon, J. Dongarra, S. Hammarling, G. Henry, A. Petitet, K. Stanley, D. Walker, and R. C. Whaley.

PARDISO in Intel® MKL is compliant with the 3.2 release of PARDISO that is freely distributed by the University of Basel. It can be obtained at http://www.pardiso-project.org.

Some FFT functions in this release of Intel® MKL have been generated by the SPIRAL software generation system (http://www.spiral.net/) under license from Carnegie Mellon University. The Authors of SPIRAL are Markus Puschel, Jose Moura, Jeremy Johnson, David Padua, Manuela Veloso, Bryan Singer, Jianxin Xiong, Franz Franchetti, Aca Gacic, Yevgen Voronenko, Kang Chen, Robert W. Johnson, and Nick Rizzolo.

6 Intel® Threading Building Blocks

For information on changes in this version of Intel® Threading Building Blocks, please read the file `CHANGES` in the TBB documentation directory.

6.1 Known Issues

Please note the following with respect to this particular release of Intel Threading Building Blocks.

6.1.1 Library Issues

- If you are using Intel Threading Building Blocks and OpenMP* constructs mixed together in rapid succession in the same program, and you are using Intel compilers for your OpenMP* code, set KMP_BLOCKTIME to a small value (e.g., 20 milliseconds) to improve performance. This setting can also be made within your OpenMP* code via the kmp_set_blocktime() library call. See the Intel compiler OpenMP* documentation for more details on KMP_BLOCKTIME and kmp_set_blocktime().
- In general, non-debug ("release") builds of applications or examples should link against the non-debug versions of the Intel Threading Building Blocks libraries, and debug builds should link against the debug versions of these libraries. On Windows systems, compile with /MD and use Intel Threading Building Blocks release libraries, or compile with /MDd and use debug libraries; not doing so may cause run-time failures. See the Tutorial in the product "Documentation" sub-directory for more details on debug vs. release libraries.

7 Disclaimer and Legal Information

Contact your local Intel sales office or your distributor to obtain the latest specifications and before placing your product order.

Copies of documents which have an order number and are referenced in this document, or other Intel literature, may be obtained by calling 1-800-548-4725, or go to: http://www.intel.com/design/literature.htm

Intel processor numbers are not a measure of performance. Processor numbers differentiate features within each processor family, not across different processor families. Go to:

http://www.intel.com/products/processor%5Fnumber/